Comparative analysis between trained from scratch, and pretrained live face recognition models, using the ResNet50 CNN architecture

Presenting:

Apostolakis Charalampos, John Theocharis

July, 2023

Outline

I. Introduction
II. Data Methodology
III. Implementation
IV. Results
V. Live Face Recognition & Demonstration
VI. Conclusions

I. Introduction (A)

- ✓ **Main Objective**: Develop accurate and robust face recognition models, comparing their capability of identifying individuals and demonstrate them using live camera face recognition
- ✓ **All models' fc layer is being transformed to identify one of 3 classes ("HarisA", "JohnT", "Unkown")**

Three models are going to be developed and assessed:

1) The 1st model is pretrained using the ResNet50 architecture and weights derived from IMAGENET1K_V2*.
   - ➢ These weights are used as a starting point for transfer learning.
   - ➢ We keep the first three out of five blocks freezed
   - ➢ The last 2 blocks and the fully connected layer are trained using our own dataset.

2) The 2nd model is deployed using the ResNet50 architecture but initialising the weights randomly (weights=None)
   - ➢ As a result, the model will start training from the beginning, using ResNet50 architecture and our own dataset

3) The 3rd model is a simple CNN architecture entirely customised and trained by ourselves.

**The ImageNet dataset is a large-scale dataset widely used for training and evaluating deep learning models, particularly for image classification tasks.*

I. Introduction (B)

*A few words about **ResNet50** architecture used among others in this project….*

→ Part of the ResNet (Residual Network) family of models and has 50 layers
→ By using these skip connections, the model can learn residual functions, which are the differences between the input and the desired output.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

→ It consists of 1 initial convolutional layer, followed by a MaxPool operation, 48 convolutional layers, and a fully connected layer after applying a global average pooling operation
→ Each convolutional layer is followed by batch normalization and ReLU activation.

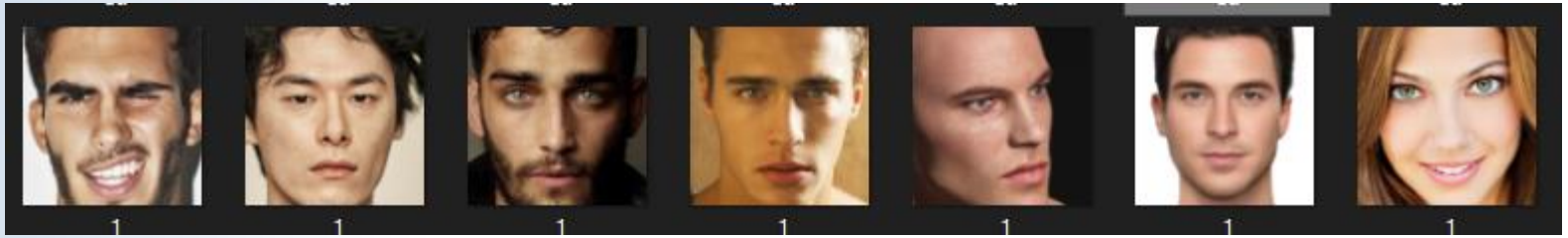
II. Data Methodology (A)

1. <u>Data Collection:</u>

In this step we collected a diverse and representative dataset of facial images. This dataset, comprised of face images, encompasses variations in lighting conditions, poses, facial expressions, ages, ethnicities, and gender.

2 sources

Kaggle website (various face images in order to train the models for the unknown class)



Our own images (HarisA, JohnT) obtained by a “face capture” script (using the OpenCV library)

- Used different cameras (3 to 4) to avoid camera bias
- Different angles in pictures and different backgrounds.

Total dataset size: ~ 20000 face images considered to be a generally medium, yet enough sized, for our low – scale project, limited to our computational resources

II. Data Methodology (A)

2. <u>Data Preprocessing and Augmentation:</u>

- Image Resize and Normalization: Resize all images to the same size and perform normalization techniques.

- Data Augmentation: Applying data augmentation methods, like random cropping, rotation, color scaling, and mirroring, to increase the dataset size and improve model generalization.

II. Data Methodology (B)

3. Model Training:

The ResNet-50 architecture and our customised CNN architecture are employed for training the face recognition models. The training process involves:

- ❖ Transfer Learning and fine tuning (1st model), valorizing the pre-trained ResNet-50 model on ImageNet
- ❖ Optimization: Tuning and selecting the appropriate hyperparameters (learning rate, batch size, etc.) to train the model.
- ❖ Loss Function: Employing a suitable loss function to optimize the model's parameters for accurate face recognition.

4. Model Evaluation:

The trained model is evaluated on the testing dataset (split the initial) to assess its performance. The evaluation metrics (i.e. accuracy, loss, F1 score) are presented in graphs to visualise the models' performance per epoch.

5. Live Demonstration, experimentation and assessment

Live Camera face recognition script is provided in order to demonstrate each model's capability to recognize correctly the faces in front of the camera (compared to models' static face images classification performance)

III. Implementation

Step 1 : Collect the necessary dataset and apply data preprocessing and transformation techniques

- **Data Collection:** We designed the "**face_capture**" script, to capture a specified number of images for multiple users' faces using a webcam and save them to individual directories (in our case we have 2 users to collect additional face frames except from "Unknown" class, pre-collected).
- **Data Preprocessing:** We developed the "**face_preprocess**" script, to parse the collected face images (jpg, png, jpeg) from the directories created with the "face_capture" script or gathered by a different source and perform face detection and preprocessing, cropping and resizing the detected faces, and normalizing their pixel values.
- **Data Augmentation:** We applied the "**augmentation_techniques**" script, to apply augmentation techniques to face images:

→ It generates augmented versions of the original face images and saves them to a new directory.

→By adding randomly generated image transformations in the training dataset, the network learns to be robust to variations in lighting conditions, contrast, and overall intensity, making it more effective at recognizing patterns in new, unseen images. The final resulting dataset can be used for training our deep learning models.

** For frontal faces detection in frames by web camera, we use the pre-trained **Haar Cascade classifier ,** part of the OpenCV library*

III. Implementation

Step 2 : Design the 3 models' architecture:

- **1st Model: ResNet50 – pretrained:** Train a CNN based on the ResNet-50 architecture and weights derived from IMAGENET1K_V2, These weights are used as a starting point.
- The first 3/5 blocks are used as such (freeze) but the last 2 blocks are trained from scratch using our own dataset
- The last fully connected layer is replaced to match the desired number of output classes. In our case 3
- the number and label of classes is determined by the number of different folders within a specified directory.
- **2nd Model: ResNet50 – not pretrained:** Train a CNN on our own dataset, based on the ResNet-50 architecture and weights initialised randomly.
  - No weights
  - all layers free.

```python
class FaceRecognitionModel(nn.Module):
    def __init__(self, num_classes):
        super(FaceRecognitionModel, self).__init__()
        # weights=ResNet50_Weights.IMAGENET1K_V2
        self.base_model = resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
        self.base_model.fc = nn.Linear(2048, num_classes)

        # freeze the first three layers
        for name, module in self.base_model.named_children():
            if name in ['layer1', 'layer2', 'layer3']:
                for param in module.parameters():
                    param.requires_grad = False

    def forward(self, x):
        features = self.base_model(x)
        return features
```

```python
    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = self.transform(image)
        label = os.path.basename(os.path.dirname(image_path))
        return image, self.classes.index(label)
```

III. Implementation

Step 2 : Design the 3 models' architecture:

- **3rd Model: Custom CNN:** Train a custom face recognition convolutional neural network (CNN) simple model, using our own created dataset with:
- 3 convolutional layers, each one followed by batch normalization, ReLU activation and max pooling (downsampling)
- Inputs images 224x224 with 3 channels (RGB) and applies filters of size 3x3 with a stride of 1 and padding of 1
- Outputs 16, 32 and 64 channels per convolutional layer respectively
- The size of the fully connected layer (fc) consists of 64 * 28 * 28 features, assuming the final feature maps have a spatial size of 28x28 after three rounds of max pooling (64 channels * [224 (initial image size) / 2^3 due to 3 times max pooling = 28] *28)

```python
class CustomModel(nn.Module):
    def __init__(self, num_classes):
        super(CustomModel, self).__init__()

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.fc = nn.Linear(64 * 28 * 28, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = nn.ReLU()(x)
        x = nn.MaxPool2d(kernel_size=2, stride=2)(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = nn.ReLU()(x)
        x = nn.MaxPool2d(kernel_size=2, stride=2)(x)

        x = self.conv3(x)
        x = self.bn3(x)
        x = nn.ReLU()(x)
        x = nn.MaxPool2d(kernel_size=2, stride=2)(x)

        x = x.view(x.size(0), -1)

        x = self.fc(x)

        return x
```

III. Implementation

Step 3 : Training, Tuning and Evaluating the 3 models

The training dataset was imported to each model, transforming the fc layer features to the number of classes (3)

```
self.base_model.fc = nn.Linear(2048, num_classes)
```

```
self.fc = nn.Linear(64 * 28 * 28, num_classes)
```

After iterative hyperparameter tuning the final parameters deemed appropriate to apply (maximising validation performance) were:

- Optimizer: Adam ("Adaptive Moment Estimation") chosen between stochastic gradient descent (SGD), AdaGrad, RMSprop
- Learning rate: 0.01 to 0.000001
- Loss function: CrossEntropyLoss
- Batch size: 32 to 128
- No of Epochs: 10 for our custom CNN (due to lighter architecture) and 5 for ResNet50
- Activation function: ReLU (Rectified Linear Unit), in all models trained
- Batch Normalization applied

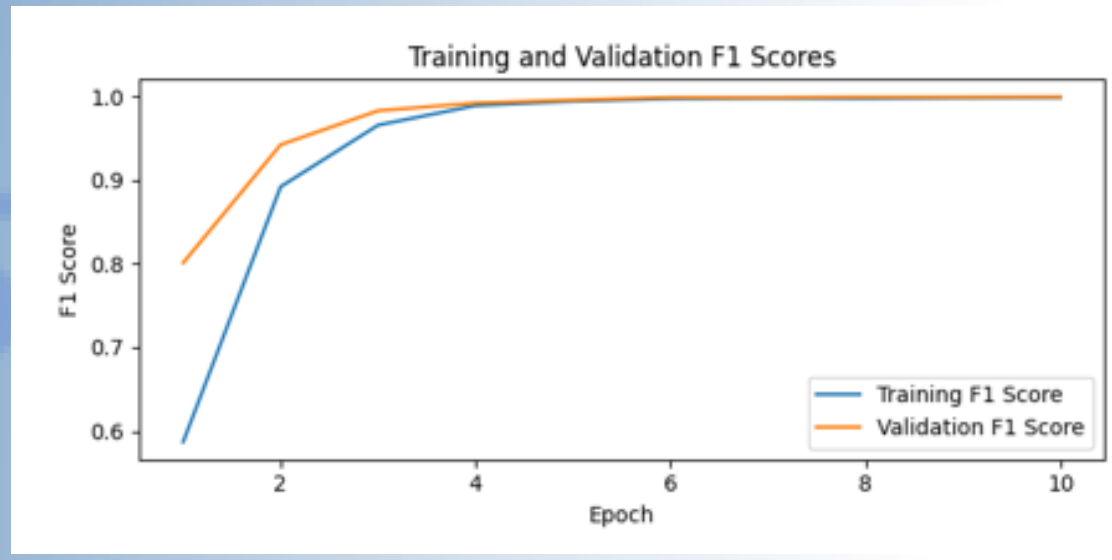
Regarding the time to train the 3 models, our records demonstrate that:
✓The 2 ResNet50 models took more than 24 hours with 5 epochs..
✓Our custom CNN model the time elapsed to train was less than 10 hours but… for 10 epochs !
✓That is more than half the time needed compared to ResNet50!!

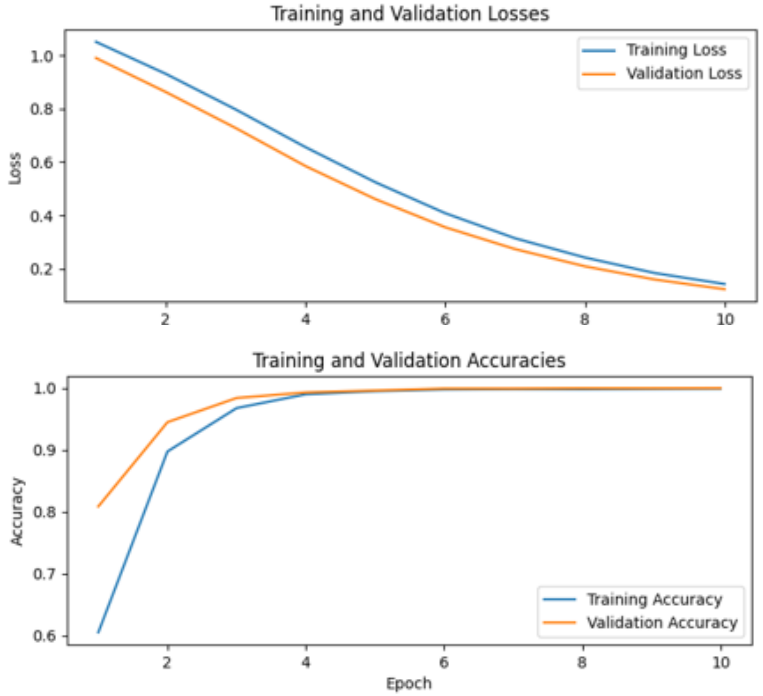
IV. Results (A)

We tried to improve the parameters combination till ending up with a high level of accuracy. All 3 models finally achieved high accuracy.

- **<u>ResNet-50 architecture and weights derived from IMAGENET1K_V2</u>**

Already since the 3rd epoch the model reaches almost 100% accuracy, however due to the small training dataset we chose to continue with 5 epochs to avoid overfitting. Here we chose to show how it behaves in 10 epochs.

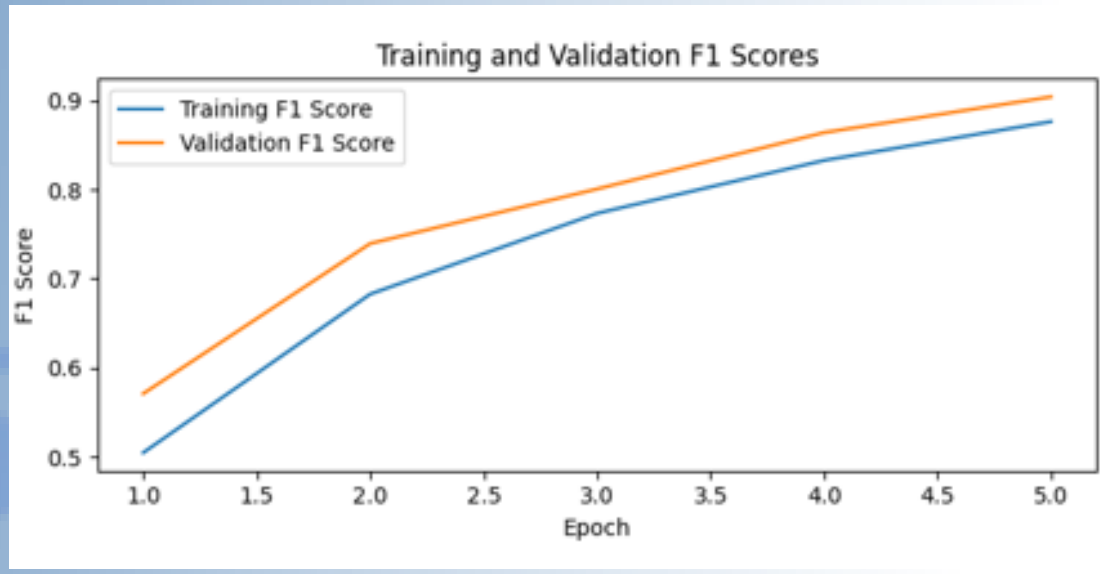


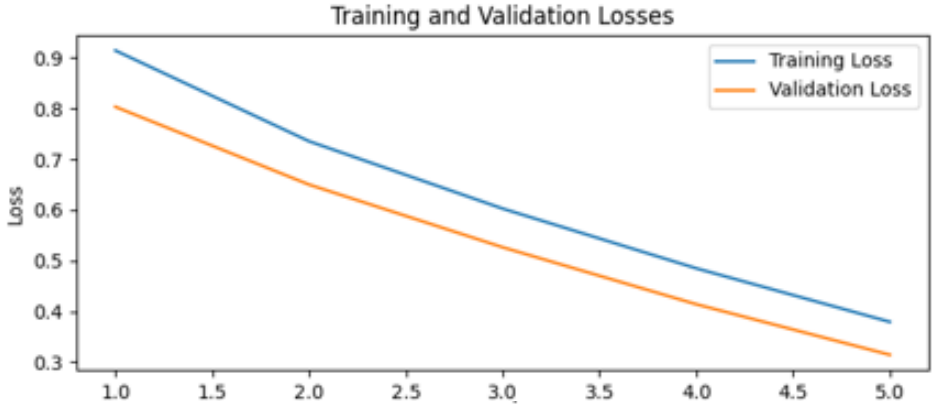


IV. Results (B)

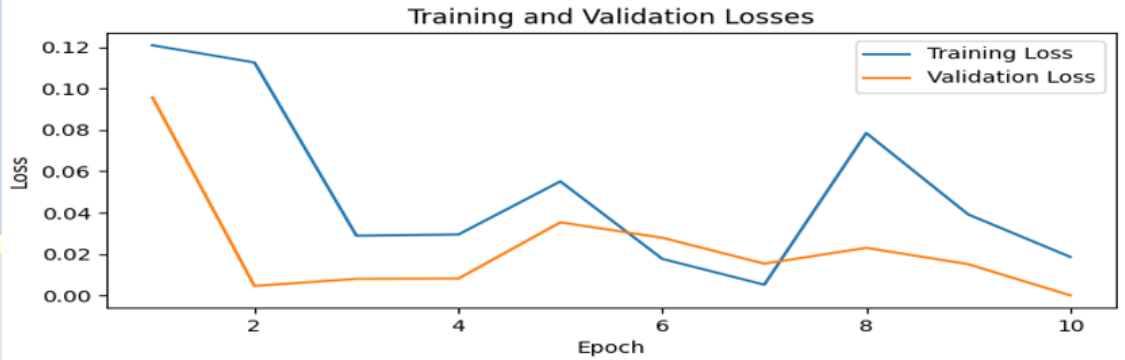
- **ResNet-50 architecture and weights = None**

Although in the training set the accuracy didn't reach maximum level, in the 5th epoch we observe a maximum accuracy and minimum loss on the validation set.

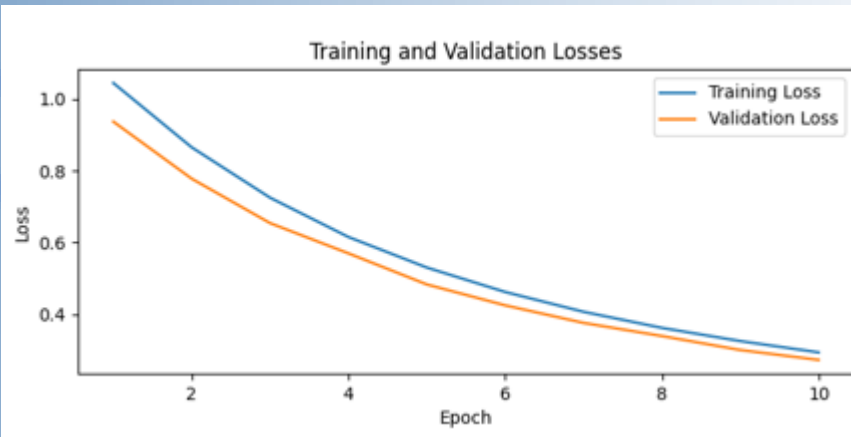


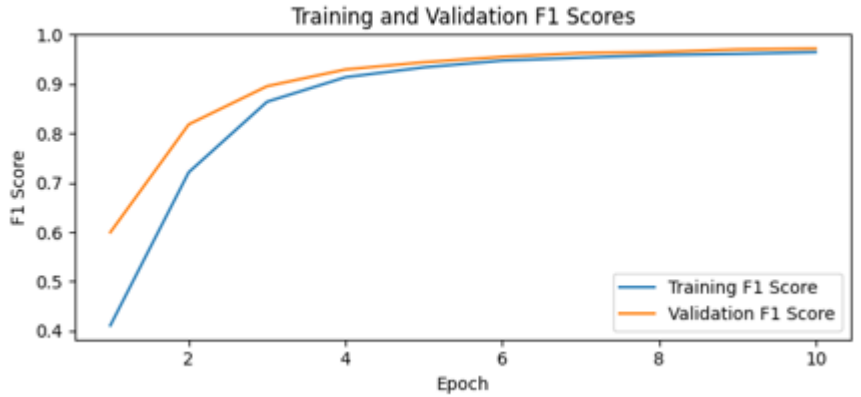


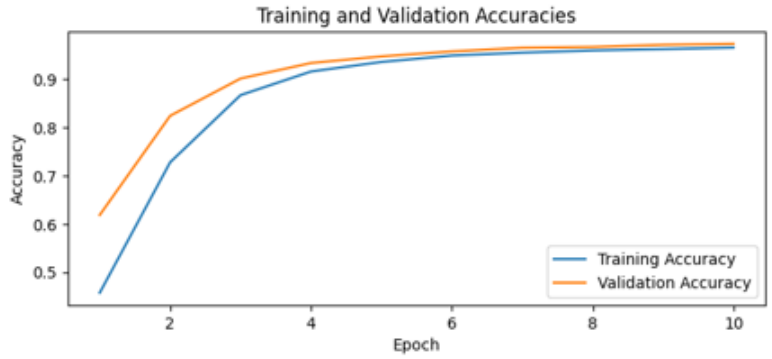
IV. Results (C)



- **<u>Custom CNN model architecture</u>**
- It is worth noting that with lr = 0.001 as applied in ResNet50 models, here we observed fluctuations in the training loss maybe
- The results after adjusting the learning rate to 0.000001 from 0.001 confirm our previous hypothesis for the learning rate. That is, now we observe a much more smooth line from epoch to epoch without fluctuations.



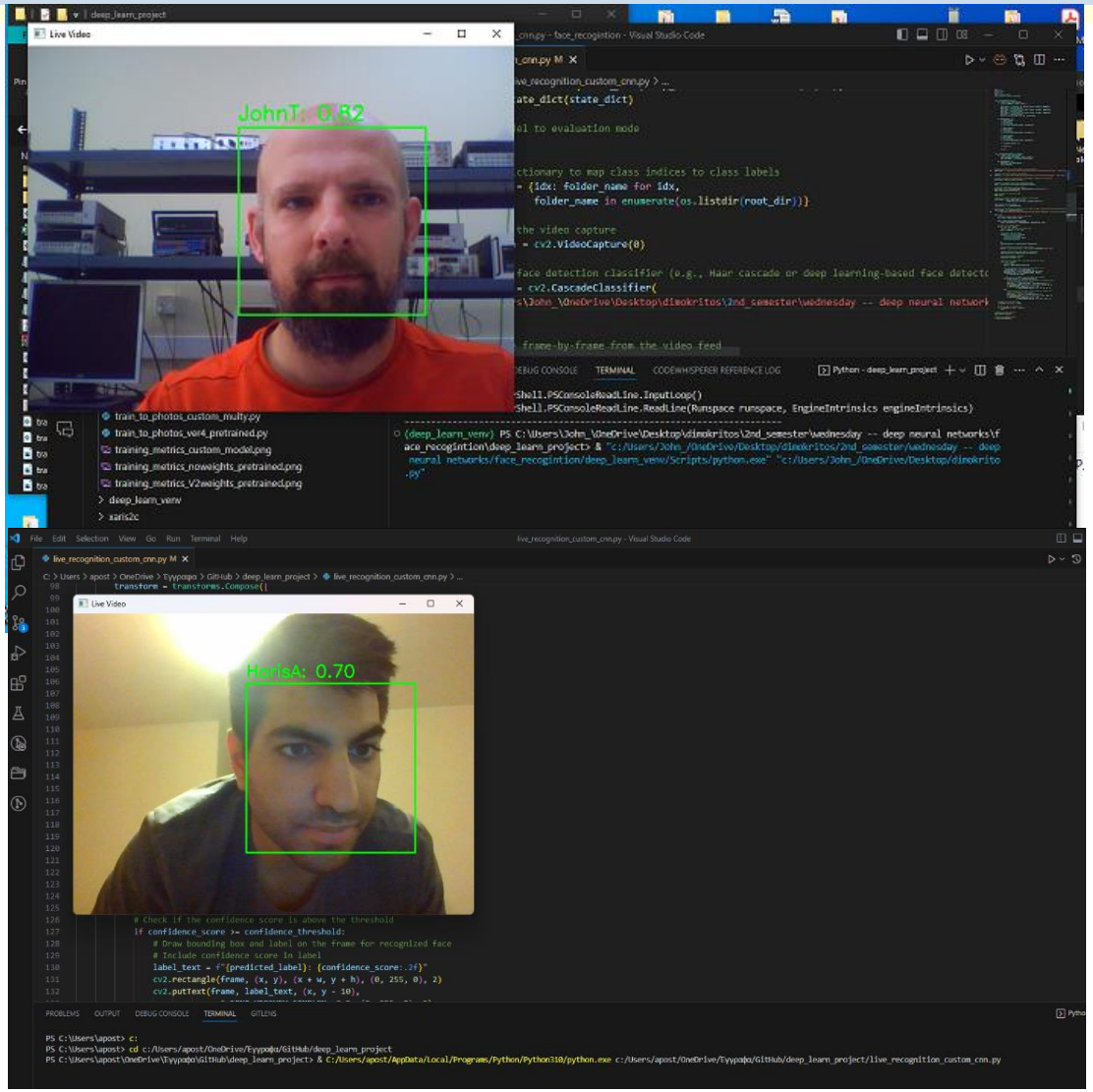




V. Live Face Recognition & Demonstration

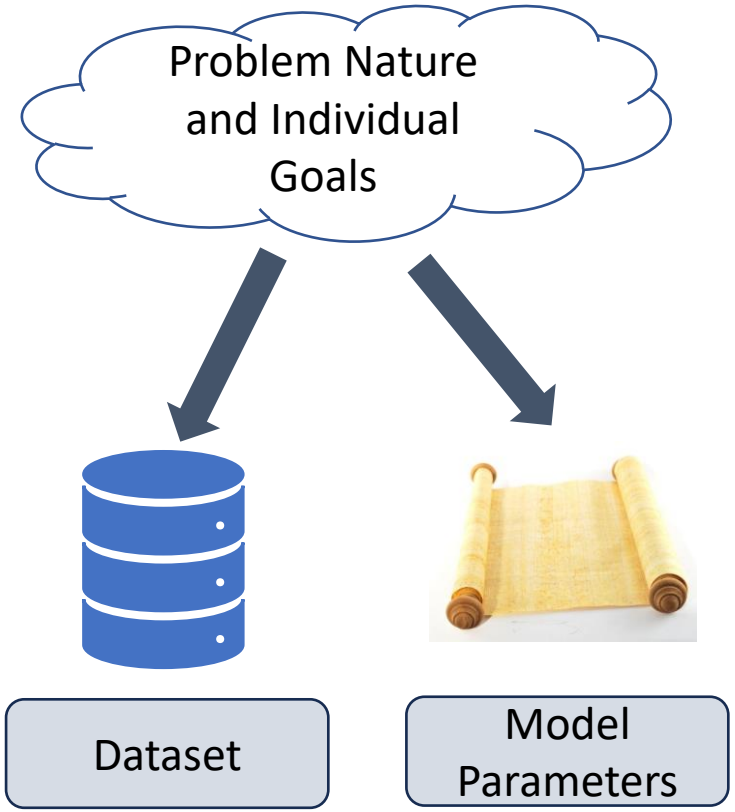
For each model we developed a script, rendering its saved model state and performing **real-time face recognition** using a webcam. This script:

- ✓ Creates an instance of the desired model architecture and loads the saved parameters
- ✓ Detects faces in the video feed, recognizes known faces and labels them with their corresponding class names. Unknown faces are labeled as "Unknown."
- ✓ Retrieves the predicted class label and confidence score for the recognized face.



VI. Conclusions (A)

- The final performance of the model depends on a combination set of many factors.
- The compound effect of many parameters as well as the difficulty and nature of the problem itself can significantly affect the performance of the model in real time.
- Due to the dynamical changes in many factors such as the faces variations, the frames background characteristics or even the camera specs, even if in static face classification the algorithm shows high accuracy, in live face recognition through camera, it can typically fail.
- The very nature of the problem and our individual goals affects the type, quality, characteristics and quantity of data needed, the cost of collecting them and developing the desired model, the accuracy that the algorithm should have (minimum confidence threshold) AND provides an intuition for the initialization of its parameters.



VI. Conclusions (B)

The parameters that jointly affect the models and their performance could be the following:

- The **number of classes** in classification problems such as in our case (face recognition). As the number of classes increases, so does the need for more data and balanced sampling to facilitate proper training of the algorithm.
- The **batch size** determines the number of samples propagated through the network before the optimizer performs a parameter update. A larger batch size may lead to more stable updates but requires more memory.
- The **Network Architecture**: A more complex architecture with a large number of parameters can potentially capture intricate patterns and achieve higher accuracy but might be prone to overfitting, especially with limited training data.
- The choice of **activation function** for each layer can influence the network's ability to model complex non-linear relationships.
- **Regularization methods** are used to prevent overfitting and improve the model's generalization. Techniques such as L1 or L2 regularization, dropout, or batch normalization can be applied .

VI. Conclusions (D)

- Parameters such as the **pooling size, stride**, and type (e.g., max pooling, average pooling) can be adjusted to control the level of spatial information preservation and generalization capacity of the network.
- The **initial values** of the network's **weights** can affect how quickly the model converges and the quality of the solutions found.
- The choice of **optimizer** affects how the model's parameters are updated during training.
- The **learning rate** determines the step size taken during parameter updates and needs to be carefully chosen to achieve a balance between convergence speed and stability (possible strategy: Learning Rate Schedule)
- The **number of epochs** determines how many times the model iterates over the entire training dataset. Too few epochs may result in underfitting, where the model hasn't learned enough from the data, leading to poor performance and vice versa. (Possible strategy: Early stopping → monitors the validation loss during training and when the validation loss starts to increase)

It often requires a combination of experience, domain knowledge, and experimentation to find the best configuration. Also don’t forget that as the complexity and problem requirements increase, so will the corresponding required resources (financing for computing power, suitable dataset etc.) to cope with them!!!

Thank you very much for your attention!

Any Questions?

True story